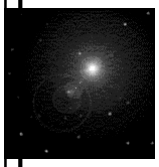


A Course on Foundations of Peer-to-Peer Systems & Applications

CS 6/75995
Foundation of Peer-to-Peer

Applications & Systems

Kent State University

Dept. of Computer Science
www.cs.kent.edu/~javed/class-P2P08/

Pedagogy

- Time Needed: 2 x 50 min classes.
- Dennis Demonstrated Actual Azurus Client (?) Download (before and after).
- Strategy
  - [why a why a seeded will stay after complete download?]
- [delete super seeding cheating part]
- Re-sequence the Fast Extension messages.
- Re-sequence the problem and solution part.
- Add few more real result graphs.
- Add analysis part (download, share ratio)

.

Vibrant Systems: BitTorrent

BitTorrent

- The protocol is the brainchild of programmer Bram Cohen, who designed it in April 2001 and released a first implementation on 2 July 2001.[1] It is now maintained by Cohen's company BitTorrent, Inc.
- One of the most significant contributor of Internet traffic.
- Is not a fully distributed P2P sharing protocol.
  - By it self is only a file distribution protocol
- Files are split into chunks (~256-512KB)
  - Chunks are not downloaded in order
  - Reassembly required after download
- Downloaders of a file *barter* for chunks of it in a tit-for-tat manner
  - Helps prevent parasitic behavior
  - Redistribute cost of upload to downloaders

FOUNDATION OF PEER-TO-PEER SYSTEMS

Overview

1. Basic Concepts
2. Basic Working Model
3. B-encoding
4. Tracker Protocols
5. Peer Wire Protocol
6. Algorithms
7. Critical Analysis

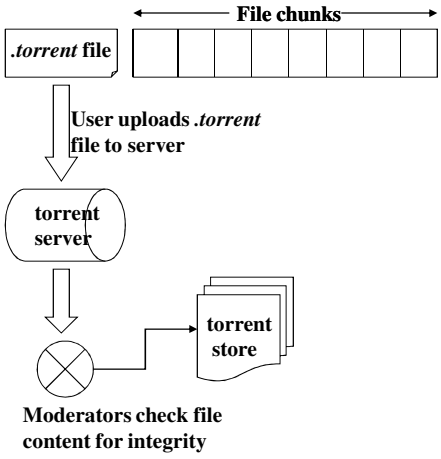
FOUNDATION OF PEER-TO-PEER SYSTEMS

Basic Concepts

- Torrent Server:
  - A regular Web Server which has a meta file (*.torrent file) containing information about a resource (file).
- Seed:
  - A peer with a complete copy of the resource and still offers it for upload (seeding)
- Leech:
  - Peer that contacts other peers to barter for chunks
- Swarm:
  - A set of peers- together, all peers downloading one resource.
- Choke:
  - A mechanism which allows client to prevent other clients from uploading to it.

Torrent Server

- Can be any ordinary web server
- Acts as a global store of *.torrent* files in the P2P network
- To reduce pollution moderators inspect content to weed out fake content
  - Moderators
  - Unmoderatted Submitters
  - Moderated Submitters



Bencoding

- Bencoding is used BitTorrent messages. It is a way to specify and organize data in a terse format. It supports the following types: byte strings, integers, lists, and dictionaries.

- **byte strings**
  - Byte strings are encoded as follows: *<string length encoded in base ten ASCII>:<string data>*
  - **Example**: *4:spam* represents the string "spam"
- **integers**
  - Integers are encoded as follows: ***i****<integer encoded in base ten ASCII>****e***
    You can have negative numbers such as ***i****-3****e***. You cannot prefix the number with a zero such as ***i****04****e***. However, ***i****0****e*** is valid.
  - **Example**: ***i****3****e*** represents the integer "3"

FOUNDATION OF PEER-TO-PEER SYSTEMS

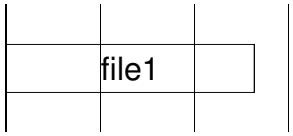
Bencoding (cont.)

- **lists**
  - Lists are encoded as follows: ***l****<bencoded values>****e***
    Lists may contain any bencoded type, including integers, strings, dictionaries, and other lists.
  - **Example:** ***l****4:spam4:eggs****e*** represents the list of two strings: ["spam", "eggs"]
- **dictionaries**
  - Dictionaries are encoded as follows:
    d*<bencoded string><bencoded element>****e***
    Keys must be bencoded strings. The values may be any bencoded type, including integers, strings, lists, and other dictionaries. Keys must be strings and appear in sorted order (sorted as raw strings, not alphanumerics). The strings should be compared using a binary comparison, not a culture-specific "natural" comparison.
  - **Example**: ***d****3:cow3:moo4:spam4:eggs****e*** represents the dictionary { "cow" => "moo", "spam" => "eggs" }
  - **Example**: ***d****4:spaml1:a1:bee*** represents the dictionary { "spam" => ["a", "b"] }
  - **Example**: ***d****9:publisher3:bob18:publisher.location4:home17:publisher-webpage15:www.example.com****e***

FOUNDATION OF PEER-TO-PEER SYSTEMS

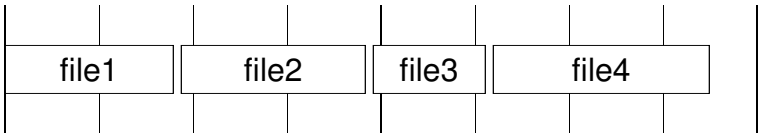
BitTorrent File Pieces

- Single File Mode



3 512K Pieces, equally dividing the file.

- Multiple File Mode



8 512K Pieces, equally dividing the files

FOUNDATION OF PEER-TO-PEER SYSTEMS

Piece Size

- The **piece length** specifies the nominal piece size, and is usually a power of 2. The piece size is typically chosen based on the total amount of file data in the torrent, constrained by the fact that piece sizes too large cause inefficiency, and too small a piece size will result in a large .torrent metadata file.
- The conventional wisdom used to be to pick the smallest piece size that results in a .torrent file no greater than approx. 50 - 75 kB (presumably to ease the load on the server hosting the torrent files). However, now that hosting storage and bandwidth are not tightly constrained, *it is best to keep the piece size to 512KB or less,* at least for torrents under 8-10GB or so, even if that results in a larger torrent file, in order to have a more efficient swarm for sharing files.
- *Every piece is of equal length except for the final piece. For multi-file case, consider the file data as one long continuous stream, composed of the concatenation of each file in the order listed in the* files *list. The pieces boundaries are then determined in the same manner as the case of a single file. Pieces may overlap file boundaries.*
- Each piece has a corresponding SHA1 hash of the data contained within that piece. These hashes are concatenated to form the *pieces* ***value in the*** **info** dictionary. This is **not** a list but rather a single string. The length of the string must be a multiple of 20.

FOUNDATION OF PEER-TO-PEER SYSTEMS

Torrent File (meta info)

- The required information about all resource (files) to be shared over BitTorrent is kept is a file called Metainfo file. This is a specially prepared file (with extension ".torrent"). It is a bencoded dictionary. It contains the keys listed below.
- **Info dictionary**: a dictionary that describes the file(s) of the torrent. There are two possible forms: one for the case of a 'single-file' torrent with no directory structure, and one for the case of a 'multi-file' torrent. (explained next).
- **announce:** The announce URL of the tracker (string). This URL gives client address of the corresponding tracker.
- **announce-list:** (optional). This optional key is used to implement lists of backup trackers.
- **creation date:** (optional) the creation time of the torrent, in standard UNIX epoch format (integer seconds since 1-Jan-1970 00:00:00 UTC)
- **comment:** (optional) free-form textual comments of the author (string)
- **created by:** (optional) name and version of the program used to create the .torrent (string)

Info	announce	announce-list (opt)	date	comment	created by

FOUNDATION OF PEER-TO-PEER SYSTEMS

Info Dictionary

- Each file is divided into pieces. These pieces are served in distributed way. The info field in torrent file contains a dictionary which contains information of all the pieces.
- **piece length:** number of bytes in each piece (integer)
- **pieces:** string consisting of the concatenation of all 20-byte SHA1 hash values, one per piece (byte string)
- **private:** (optional) If it is set to "1", the client MUST publish its presence to get other peers ONLY via the trackers explicitly described in the metainfo file. If this field is set to "0" or is not present, the client may obtain peer from other means, e.g. PEX peer exchange, dht. Here, "private" may be read as "no external peer source".
- There are two modes "single file" and "multiple file". (more fields based on file mode..)

FOUNDATION OF PEER-TO-PEER SYSTEMS

Info Dictionary (cont..)

+(Info in Single File Mode:)

- **name:** the filename of the file. This is purely advisory. (string)
- **length:** length of the file in bytes (integer)
- **md5sum:** (opt.) a 32-character hexadecimal string the MD5 sum of the file.

OR

+(Info in Multiple File Mode)

- **name:** the filename of the directory in which to store all the files. This is purely advisory. (string)
- **files:** a list of dictionaries, one for each file. Each dictionary in this list contains the following keys:
  - **length:** length of the file in bytes (integer)
  - **md5sum**: (optional) a 32-character hexadecimal string MD5 sum of the file.
  - **path:** a list. Each element in the list corresponds to either a directory name or (in the case of the final element) the filename. For example, a the file "dir1/dir2/file.ext" would consist of three string elements: "dir1", "dir2", and "file.ext". This is encoded as a bencoded list of strings such as *l''***4**:dir1'**4**:*dir2*'**8**:file.ext'**e***

BitTorrent:

Contacting Tracker

Tracker HTTP/ HTTPS Protocol

- The tracker is an HTTP/HTTPS service which responds to HTTP GET requests.
- The requests include metrics from clients that help the tracker keep overall statistics about the torrent. The response includes a peer list that helps the client participate in the torrent.
- The base URL consists of the "announce URL" as defined in the metadata (.torrent) file. The parameters are then added to this URL, using standard CGI methods (i.e. a '?' after the announce URL, followed by 'param=value' sequences separated by '&').

FOUNDATION OF PEER-TO-PEER SYSTEMS

Tracker Request Parameters

Each client must send following parameters to tracker via GET request.

- **info_hash**: urlencoded 20-byte SHA1 hash of the *value* of the *info* key from the Metainfo file. Note that the *value* will be a bencoded dictionary. [apparently client can start with info_hash of any piece and get directed into the swarm- javed]
- **peer_id:** urlencoded 20-byte string used as a unique ID for the client, generated by the client at startup. This is allowed to be any value, and may be binary data.
- **port**: The port number that the client is listening on. Ports reserved for BitTorrent are typically 6881-6889. Clients may choose to give up if it cannot establish a port within this range.
- **uploaded**: The total bytes amount uploaded (since the client sent the 'started' event to the tracker) in base ten ASCII.
- **downloaded**: The total bytes downloaded (since the client sent the 'started' event to the tracker) in base ten ASCII.

FOUNDATION OF PEER-TO-PEER SYSTEMS

Tracker Request Parameters (cont..)

- **left**: The number of bytes this client still has to download.
- **compact**: Indicates the client accepts a compact response. The peers list is replaced by a peers string with 6 bytes per peer. Some trackers only support compact responses .
- **no_peer_id**: Indicates that the tracker can omit peer id field in peers dictionary. This option is ignored if compact is enabled.
- **event**: If specified, must be one of *started*, *completed*, *stopped*, (or empty which is the same as not being specified). If not specified, then this request is one performed at regular intervals.
  - **started**: The first request to the tracker *must* specify this value.
  - **stopped**: Must be sent to the tracker if the client is shutting down gracefully.
  - **completed**: Must be sent to the tracker when the download completes. However, must not be sent if the download was already 100% complete when the client started. This is to allow the tracker to increment the "completed downloads" metric based solely on this event.

Tracker Request Parameters (cont..)

- **IP:** (optional) The true IP address of the client machine, in dotted quad format or rfc3513 defined hexed IPv6 address.
  - *Not necessary if the address of the client can be determined from the IP address from which the HTTP request came. The parameter is only needed in the case if the client is communicating to the tracker through a proxy (or a transparent web proxy/cache.), or on the same local side of a NAT gateway.*
- **numwant**: (optional) Number of peers that the client would like to receive from the tracker. This value is permitted to be zero. If omitted, typically defaults to 50 peers.
- **key**: (optional) An additional identification that is not shared with any users. It is intended to allow a client to prove their identity should their IP address change.
- **trackerid**: (optional) If a previous announce contained a tracker id, it should be set here.

Response from Tracker

The tracker responds with "text/plain" document consisting of a bencoded dictionary with the following keys to receive a list of peers:

- **failure reason**: If present, then no other keys may be present. A human-readable error message as to why the request failed (string).
- **warning message**: (new, optional) Similar to failure reason, but the response still gets processed normally. The warning message is shown just like an error.
- **interval**: Interval in seconds that the client should wait between sending regular requests to the tracker
- **min interval**: (optional) Minimum announce interval. If present clients must not reannounce more frequently than this.
- **tracker id**: A string that the client should send back on its next announcements. If absent and a previous announce sent a tracker id, do not discard the old value; keep using it.
- **complete**: number of peers with the entire file, i.e. seeders (integer)
- **incomplete**: number of non-seeder peers, aka "leechers" (integer).

Response from Tracker (cont..)

- **peers**: (dictionary model) The value is a list of dictionaries, each with keys:
  - **peer id**: peer's self-selected ID, as described above for the tracker request (string)
  - **ip**: peer's IP address either IPv6 (hexed) or IPv4 (dotted quad) or DNS (string)
  - **port**: peer's port number (integer)
- **peers**: (compact model) Instead of using the dictionary model described above,
  - The **peers** value may be a string consisting of multiples of 6 bytes. First 4 bytes are the IP address and last 2 bytes are the port number.
  - The list of peers is length 50 by default. If there are fewer peers in the torrent, then the list will be smaller. Otherwise, the tracker randomly selects peers to include in the response.
  - *The tracker may choose any other intelligent mechanism for peer selection. For instance, reporting seeds to other seeders could be avoided.*
- *Clients may send a request more often than the specified interval, if an event occurs (i.e. stopped or completed) or if the client needs to learn about more peers. However, it is considered bad practice to "hammer" on a tracker to get multiple peers. If a client wants a large peer list in the response, then it should specify the **numwant** parameter.*

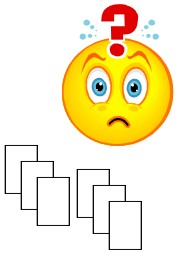
BitTorrent:
Peer-to-Peer Piece Exchange

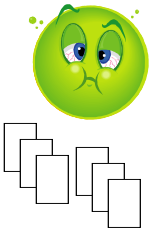
Peer Wire Protocol (TCP)

- **The peer protocol facilitates the exchange of pieces as described in the '*metainfo* file.**
- The peer wire protocol consists of an initial handshake. After that, peers communicate via an exchange of length-prefixed messages.

Autonomy & Cooperation

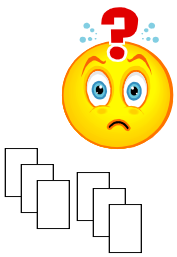
- A client has no means to force a remote peer to contribute a piece. It need to elicit cooperation. Fortunately same is true about the remote peer (if it not a seed already). Therefore, a remote peer also needs cooperation.

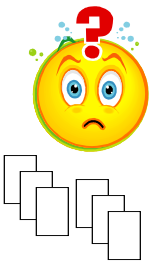




Autonomy & Cooperation

- A client has no means to force a remote peer to contribute a piece. It need to elicit cooperation. Fortunately same is true about the remote peer (if it not a seed already). Therefore, a remote peer also needs cooperation.

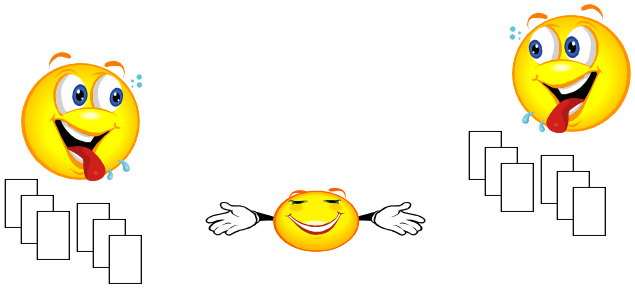




FOUNDATION OF PEER-TO-PEER SYSTEMS

Choke & Interest

- BitTorrent suggest to encourage this mutual cooperation by exchanging two state information. Each client assesses the intentions to “choke” and “receive”.



FOUNDATION OF PEER-TO-PEER SYSTEMS

Choke & Interest

- **Choked**:
  - Whether or not the remote peer has choked this client. When a peer chokes the client, it is a notification that no requests will be answered until the client is unchoked. The client should not attempt to send requests for blocks, and it should consider all pending (unanswered) requests to be discarded by the remote peer.
- **Interested**:
  - Whether or not the remote peer is interested in something this client has to offer. This is a notification that the remote peer will begin requesting blocks when the client unchokes them.

FOUNDATION OF PEER-TO-PEER SYSTEMS

Choke & Interest

- *Client will also need to keep track of whether or not it is interested in the remote peer, and if it has the remote peer choked or unchoked.*
- So, the real list looks something like this:
  - **am_choking**: this client is choking the peer
  - **am_interested**: this client is interested in the peer
  - **peer_choking**: peer is choking this client
  - **peer_interested**: peer is interested in this client

FOUNDATION OF PEER-TO-PEER SYSTEMS

Choke & Interest

- Client connections start out as "choked" and "not interested". In other words:
  - **am_choking** = 1
  - **am_interested** = 0
  - **peer_choking** = 1
  - **peer_interested** = 0
- A block is downloaded by the client when the client is interested in a peer, and that peer is not choking the client. A block is uploaded by a client when the client is not choking a peer, and that peer is interested in the client.
- It is important for the client to keep its peers informed as to whether or not it is interested in them. This state information should be kept up-to-date with each peer even when the client is choked. This will allow peers to know if the client will begin downloading when it is unchoked (and vice-versa).

FOUNDATION OF PEER-TO-PEER SYSTEMS

Handshake Message

- **handshake** (49+len(pstr)) bytes *: <pstrlen><pstr><reserved><info_hash><peer_id>*
  - **pstrlen**: string length of <pstr>, as a single raw byte
  - **pstr**: string identifier of the protocol
  - **reserved**: 8 bytes. All current implementations use all zeroes.
  - **info_hash**: 20-byte SHA1 hash of the info key in the metainfo file and transmitted in tracker requests.
  - **peer_id**: 20-byte string used as a unique ID for the client also transmitted in tracker requests.
- In version 1.0 of the BitTorrent protocol, pstrlen = 19, and pstr = "BitTorrent protocol".

FOUNDATION OF PEER-TO-PEER SYSTEMS

Handshaking

- The initiator of a connection is expected to transmit their handshake immediately after opening a connection.
- The recipient may wait for the initiator's handshake, if it is capable of serving multiple torrents simultaneously (torrents are uniquely identified by their ***info_hash****).*
- However, the recipient must respond [with another handshake-javed] as soon as it sees the ***info_hash*** part of the handshake.
- If a client receives a handshake with an ***info_hash*** that it is not currently serving, then the client must drop the connection.
- Also, if the initiator of the connection receives a handshake in which the ***peer_id*** does not match the expected ***peer_id,*** then the initiator is expected to drop the connection. *(*The ***peer_id*** from the tracker and in the handshake are expected to match.)

FOUNDATION OF PEER-TO-PEER SYSTEMS

Bitfield Message

- **bitfield: <len=0001+X><id=5><bitfield>**
- The ***bitfield*** message may only be sent immediately after the handshaking sequence and before any other messages are sent. It need not be sent if a client has no pieces.
- The payload is a x bit ***bitfield*** representing the pieces that have been successfully downloaded. The high bit in the first byte corresponds to piece index 0. Bits that are cleared indicated a missing piece, and set bits indicate an available piece. Spare bits at the end are set to zero.
- *A **bitfield** of the wrong length is considered an error. Clients should drop the connection if they receive **bitfields** that are not of the correct size, or if the **bitfield** has any of the spare bits set.*

Runtime Messages

- All of the remaining messages in the protocol take the form of <length prefix><message ID><payload>. The length prefix is a four byte big-endian value. The message ID is a single decimal character. The payload is message dependent.
- **keep-alive:** <len=0000>
  - Peers may close a connection if they receive no messages (keep-alive or any other message) for a certain period of time. This amount of time is generally two minutes.

The following messages has no payload

- **choke:** <len=0001><id=0>
- **unchoke:** <len=0001><id=1>
- **interested:** <len=0001><id=2>
- **not interested**: <len=0001><id=3>

Runtime Messages

- **have: <len=0005><id=4><piece index>**
- It is used to dynamically notify others about new pieces that has just been successfully downloaded and verified via the hash. The payload is the index of a piece.
- Can we save some Have messages ("have Suppression") ?
  - *Because peers are extremely unlikely to download pieces that they already have, a peer may choose not to advertise having a piece to a peer that already has that piece.*
  - *Such "HAVE suppression" will result in a 50% reduction in the number of HAVE messages, this translates to around a 25-35% reduction in protocol overhead.*
  - *However, it may be worthwhile to send a HAVE message to a peer that has that piece already since it will be useful in determining which piece is rare.*
- *A malicious peer might also choose to advertise having pieces that it knows the peer will never download. Due to this attempting to model peers using this information is a **bad idea**.*

FOUNDATION OF PEER-TO-PEER SYSTEMS

Runtime Messages

- **request: <len=0013><id=6><index><begin><length>**
- The **request** message is fixed length, and is used to request a block. The payload contains the following information:
  - **index**: zero-based piece index
  - **begin**: zero-based byte offset within the piece
  - **length**: requested length (bytes).
- A client may close connections if request amount is over a threshold (such as greater than 128KB).

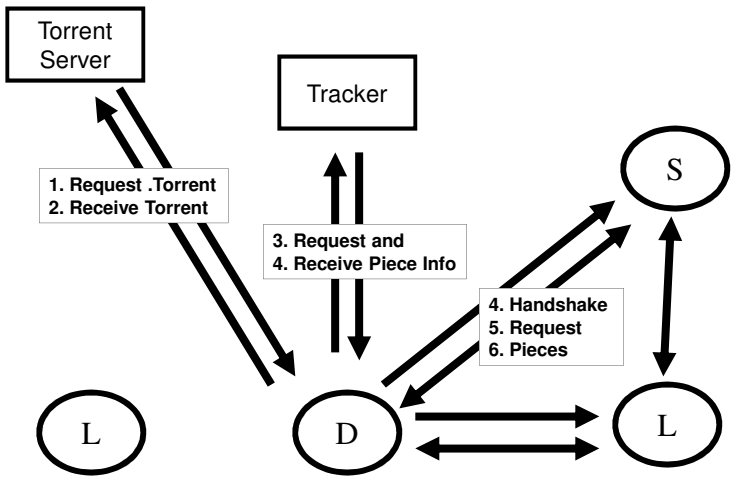
FOUNDATION OF PEER-TO-PEER SYSTEMS

Runtime Messages

- piece: <len=0009+X><id=7><index><begin><block>
- In response to a ***request***, when a peer want to sent the block it send it in piece message along with following information:
  - index: integer specifying the zero-based piece index
  - begin: integer specifying the zero-based byte offset within the piece
  - block: block of data, which is a subset of the piece specified by index.
- cancel: <len=0013><id<=8><index><begin><length>
- The cancel message used to cancel block requests. It's fields are identical to request message. It is typically used during "End Game" (see the Strategy section).
- port: <len=0003><id=9><listen-port>
- The port message is sent by newer versions of the Mainline that implements a DHT tracker. The listen port is the port this peer's DHT node is listening on. This peer should be inserted in the local routing table (if DHT tracker is supported).

FOUNDATION OF PEER-TO-PEER SYSTEMS

BitTorrent Swarm



FOUNDATION OF PEER-TO-PEER SYSTEMS